Ptr-type and Read-type routines

Wed, Feb 22, 1995

Data request support for local stations/IRMs uses "ptr-type" routines to "compile" a request for data specified via a listype#, where the request in this context is comprised of an array of "idents." A ptr-type routine scans the array of idents and produces an array of "internal ptrs" that represents a kind of "object code" for the original request. This is done so that "read-type" routines can generate the reply data most efficiently, especially for the case of requests for periodic replies.

To illustrate this simply, consider a request for readings of analog channels. The listype# for analog readings is 0. The idents used for such a request are channel numbers, each given as a two-word structure, the first word being a node# and the second word a channel#. The analog data pool of a local station is organized as a simple array of records, one field of such records being used for the reading word value. The format of internal ptr used in this case is a pointer to the given channel's reading field in its analog data pool record. The update logic, which utilizes a read-type routine, is a simple loop that gets an internal ptr from the array, de-references it to obtain the reading value result, and loops over the number of idents—the same as the number of internal ptrs—presented in the original request.

For the Classic protocol, as used by local or page applications, a data request is made specifying an array of listypes and an array of idents. If more than one listype is used, each must be associated with the same ident types, as the same ident array is "compiled" into internal ptrs for each listype given in the request. Sometime after the request has been made, the application invokes `Collect` to retrieve the results. Typically, an application does this on the next operating cycle. If the data should not yet be available, because the request required data to be retrieved from another node, then the `Collect` routine waits for the external data to arrive. There is a time-out for this that is normally 50 ms after the current cycle.

There are only a few error return codes that `Collect` returns, as follows:

0	No errors.
1	Invalid list#. Maybe the request was really strange.
2	Data not yet available. `Collect` called on same cycle request was made.
3	Internal corruption of request context.
4	Bus error detected in collecting answers.
5	Too few bytes received from an external node
6	Too many bytes received from an external node
7	Answers tardy, but have received answers at least once from ext node.
8	Answers tardy. Nothing ever received from ext node.

Note that the only nonzero error codes normally seen are 4, 7, 8. Ask for memory data using an address ident that is invalid for that station's hardware, and an error 4 is returned. Ask for data from an invalid node# or a valid, but non-operating node#, and an 8 will be returned. Ask for data from a node that is operating at a rate slower than the requesting node, and intermittent 7 errors will appear. During a periodic request activity, if that node drops off the network, solid 7 errors will occur.

The Ptr-type routines that build internal ptrs try to do so without error. If a system table does not exist, for example, the request cannot reasonably be supported by this node, so zeros are returned for the answer to the request. The ptr-type routine and the corresponding read-type routine for the given listype work together to produce the answer results. The internal ptr is the output of the ptr-type routine and the input of the read-type routine. There is no mechanism for returning an error code in the call to the data request procedure. The only error return that a read-type routine can return to `Collect` is a bus error indicator, which causes `Collect` to return error code 4. One might say that the attitude here is that a faulty request results in answer data of zeros, by definition.

In writing a ptr-type routine, consideration must be given to what is needed by the corresponding read-type routine to define the answer data. In the case of an ident that is from an external node, an external answer ptr form is used that is usually a ptr to the proper place in that external node's dedicated reply buffer for that request, which is allocated as part of the request block data structure for that request. In order to indicate that case of an internal ptr value, the sign bit is set in the internal ptr, which is a 32-bit longword. This means that typically one will find in a read-type routine corresponding logic that detects the presence of the sign bit to signal an external ptr case, removes the sign bit, then uses the low 31 bits as a ptr into the proper place in the external node's reply buffer in the request block. (All dynamic memory, and hence all request blocks, reside at memory addresses below 1 MB, so nothing is lost by commandeering bit# 31 for this purpose.

The case of a memory address ident is special. In this case, the user specifies an arbitrary 32-bit memory address from which data is to be retrieved. So we must use all 32 bits of the internal ptr for the given memory address. So in this case, the ptr-type routine does *not* set bit# 31 of the internal ptr to mark the external answer ptr case. And the read-type routine merely copies memory data from the address given. If that address happens to point to the external answer buffer, memory is copied from that buffer. If it happens to point to a field in a system table entry, memory is copied from there. In either case, memory is copied to produce the answers. It does not matter from whence it is copied.